Modern Engineering

January 17 | Day 6

Prudential Financial

GENERAL ASSEMBLY

LESSON ROADMAP

Introducing React

React Installation

Thinking in Components

Passing Data with Props

Blog Post Lab

MEF MODULE 3 DAY 6: React Foundations

Schedule	
9:00–9:15 am	Welcome and Warm-Up
9:15–10:30 am	Introduction to React, React Components
10:30 - 10:45 am	Hello Component Walkthrough
10:45 - 12:30 pm	Passing Data with Props
12:30–1:30 pm	**Lunch**
1:30-2:30 pm	**Nesting Components**
2:30–4:50 pm	**Lord Of The Rings Exercise**
4:50–5:00 pm	Bring It Home

LEARNING OBJECTIVES

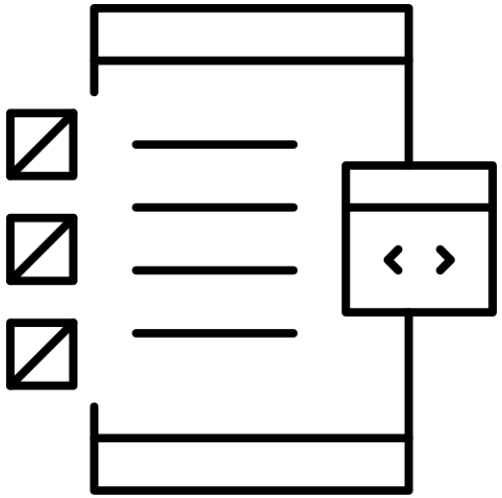
1. Think in react. Understand the fundamentals of React and its place among Single Page Application tools.
2. Use a build tool to bootstrap a full React application.
3. Use **components** to break down an application's UI into modular parts.
4. Pass data between components using **props**.

Modern Engineering

Introduction to React

What is React JS?

1. Let's think about where you might see a React.js app?



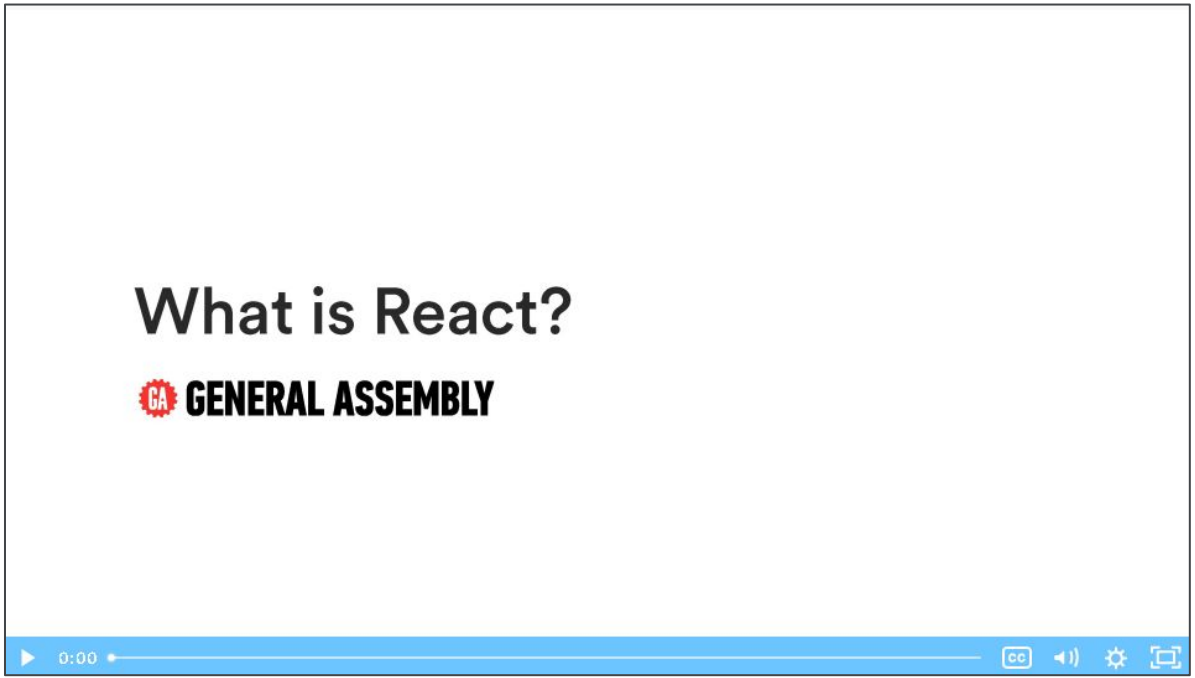
What is React JS?

2 **Let’s look at some examples:**

Facebook actually built React! It needed web pages that could change quickly based on a user's interaction — your Facebook feed, for example.

Instagram public feed and internal system are entirely built on React.

What is React JS?



3

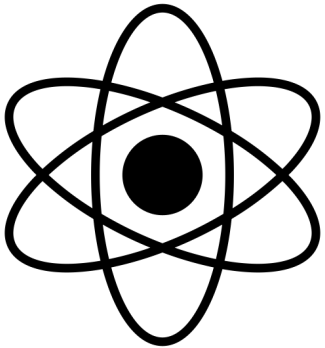
Let's watch a video

Have you ever wondered about the history behind React?

What is React JS?

React is a JavaScript library for building user interfaces.

- It uses a **virtual DOM** which optimizes the performance of updates to the user interface
- It follows a more flexible, composable approach to building web applications, whereas Angular has a more opinionated and prescriptive approach.
- **React uses JSX**, a syntax extension for JavaScript that allows mixing HTML and JavaScript in one file
- It primarily focused on building the view layer of web applications



How is React similar to Angular?

React & Angular

- Both are open-source and actively maintained by their respective communities.
- Both use a **component-based architecture** to build web applications, which allows for reusable and modular code.
- Both are supported by a large ecosystem of libraries and tools.

Key Angular Features

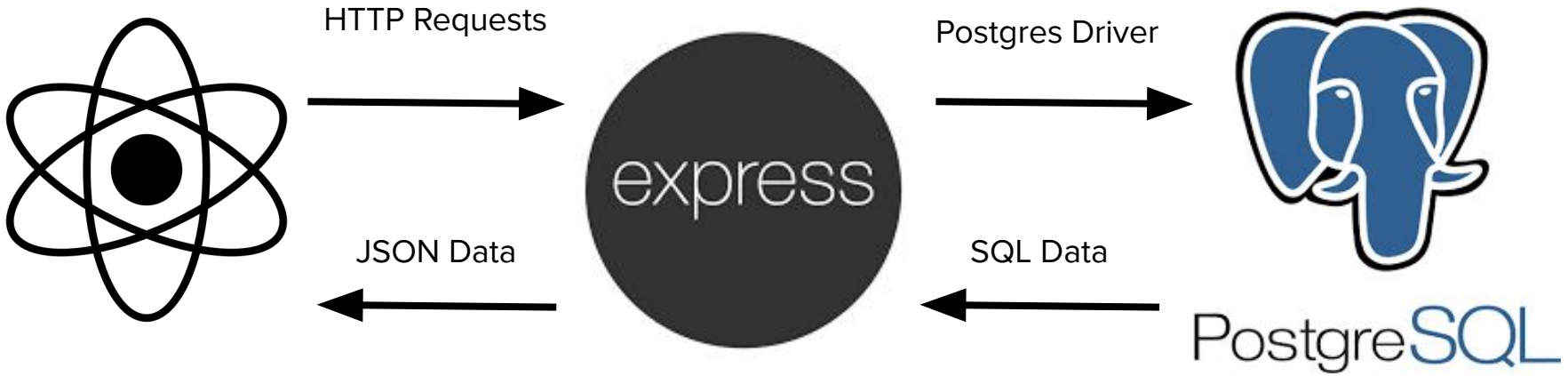
It's a **full-featured framework** that includes a lot of additional features such as dependency injection, routing, and a powerful template system.

Uses a **more traditional** two-way data binding model.

Uses **template languages** such as HTML and TypeScript

It provides a more comprehensive solution for building web applications**.**

Decoupling the Front End from the Back End

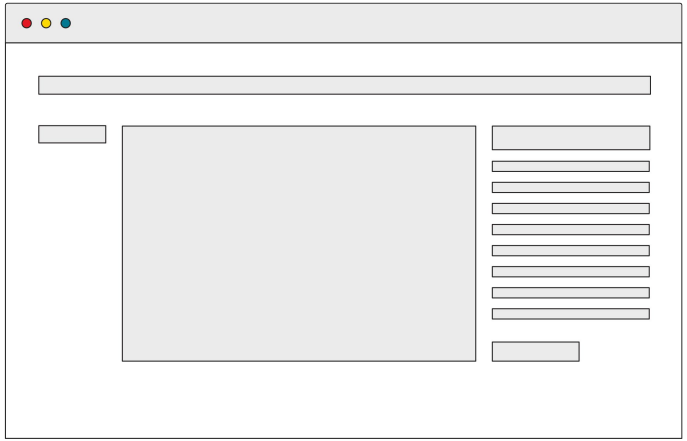


Front-End Review: The DOM

Browsers read your HTML and create an object in the computer's memory for each part. That HTML layout is called a "data model" because it describes the structure of your webpage.

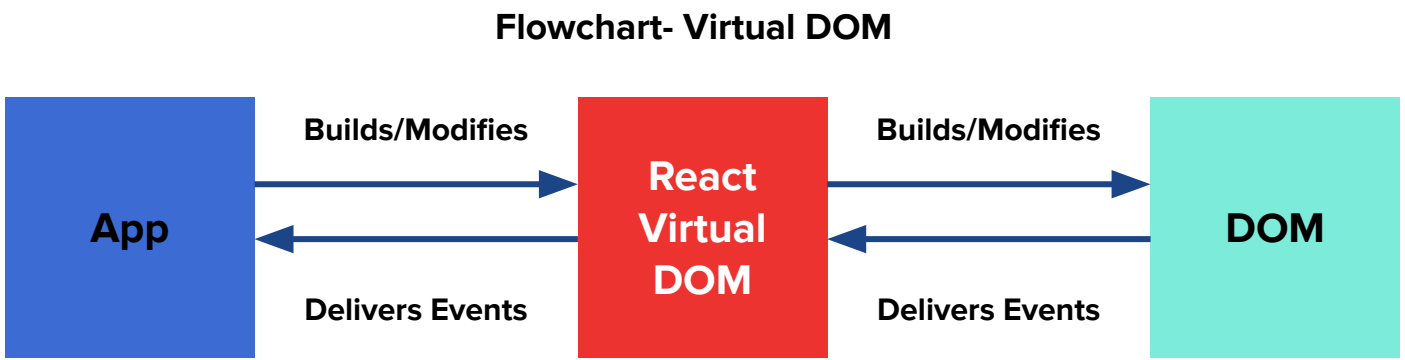
The **Document Object Model** (DOM) is the browser's JavaScript representation of your HTML elements.

Javascript attaches **Event Listeners** to DOM elements to power interactivity.



Virtual Dom

- VirtualDOM: a concept used in React to optimize the performance of updates to the user interface.
  - React apps give a smooth user experience by avoiding full refreshes, instead only updating small pieces of the site in response to user actions



What is React in terms of code?

React components commonly use **JSX**, a unique mix of HTML and JavaScript

Source: simplilearn.com

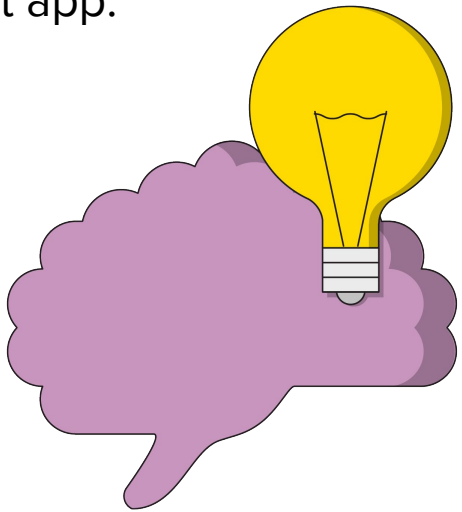
Guided Walk-Through:

React Installation with Create React App

15 minutes

We'll use a standard react build tool to spin up a new react app:

```
npx create-react-app hello-react
```



Let's explore what we get!

cd hello-react
code .
npm run start

Modern Engineering

React Components

React Components

In this app example, you'll see that there are five components. The numbers in the image correspond to the numbers below.

1. **FilterableProductTable (orange):** contains the entirety of the example
2. **SearchBar (blue):** receives all *user input*
3. **ProductTable (green):** displays and filters the *data collection* based on *user input*
4. **ProductCategoryRow (turquoise):** displays a heading for each *category*
5. **ProductRow (red):** displays a row for each *product*

Source: Thinking in React-React

Planning and Creating React Components

There are **5 Steps** for building a dynamic component using React.

Step 1

Break the UI into a complete hierarchy

Step 2

Build a static version in React

Step 3

Identify the minimal representation of UI state

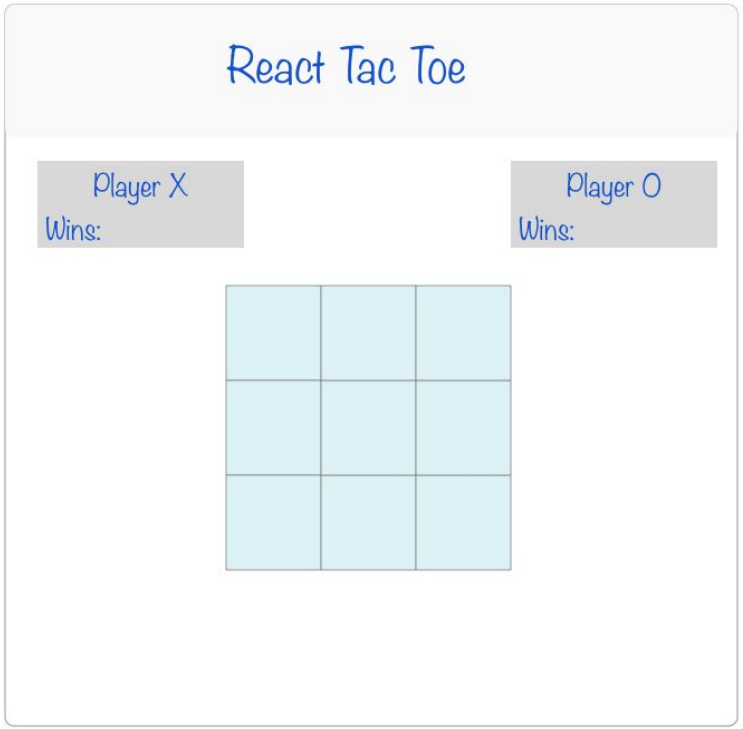
Step 4

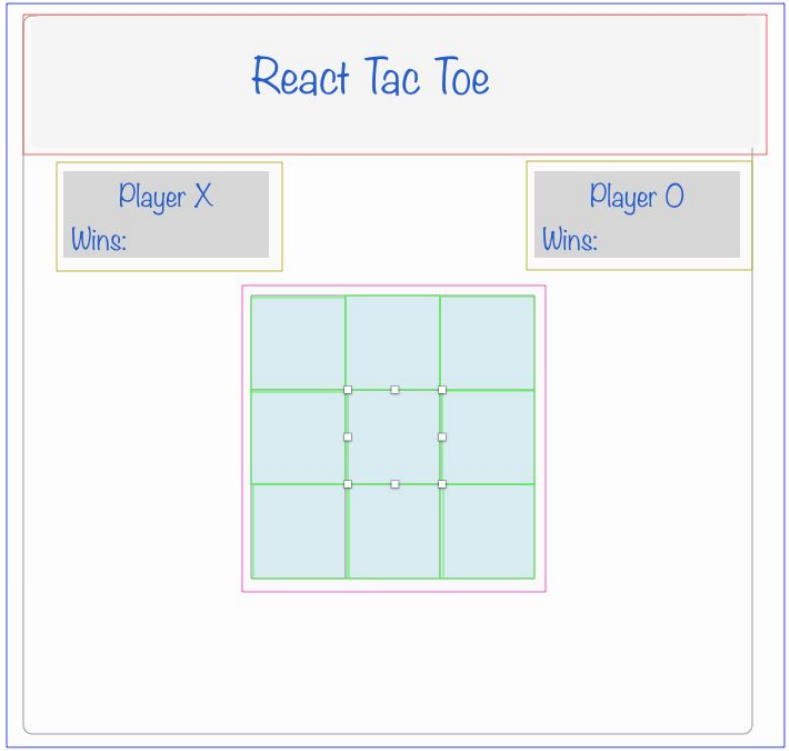
Identify where your state should live

Step 5

Add data flow

React Components: Build a Static Version in React





Functional Components

A **functional component** is defined as a JavaScript function that returns JSX markup.

Note: They do not have an internal state and do not have lifecycle methods but can use **hooks** to manage state and lifecycle methods.

Let's look at two examples

```
1  function Welcome(props) {
2      return <h1>Hello, {props.name}</h1>;
3  }
```

```
1  const Welcome = (props) => {
2      return <h1>Hello, {props.name}</h1>;
3  }
```


Research Alert: Class Components

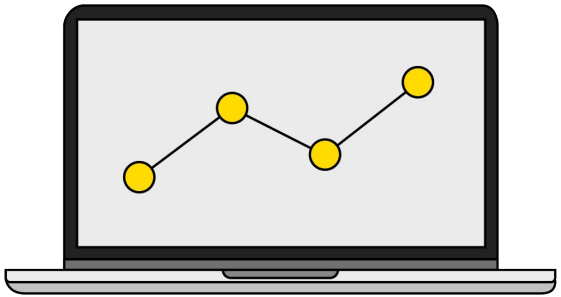
Class components are the original, legacy way of defining React components using an ES6 class. They have their own internal state and lifecycle methods, and can be used to perform specific actions during the component lifecycle.

We will use functional components exclusively, so be careful when googling! Plenty of StackOverflow responses will be about Class components.

```
1  class Welcome extends React.Component {
2    render() {
3      return <h1>Hello, {this.props.name}</h1>;
4    }
5  }
```

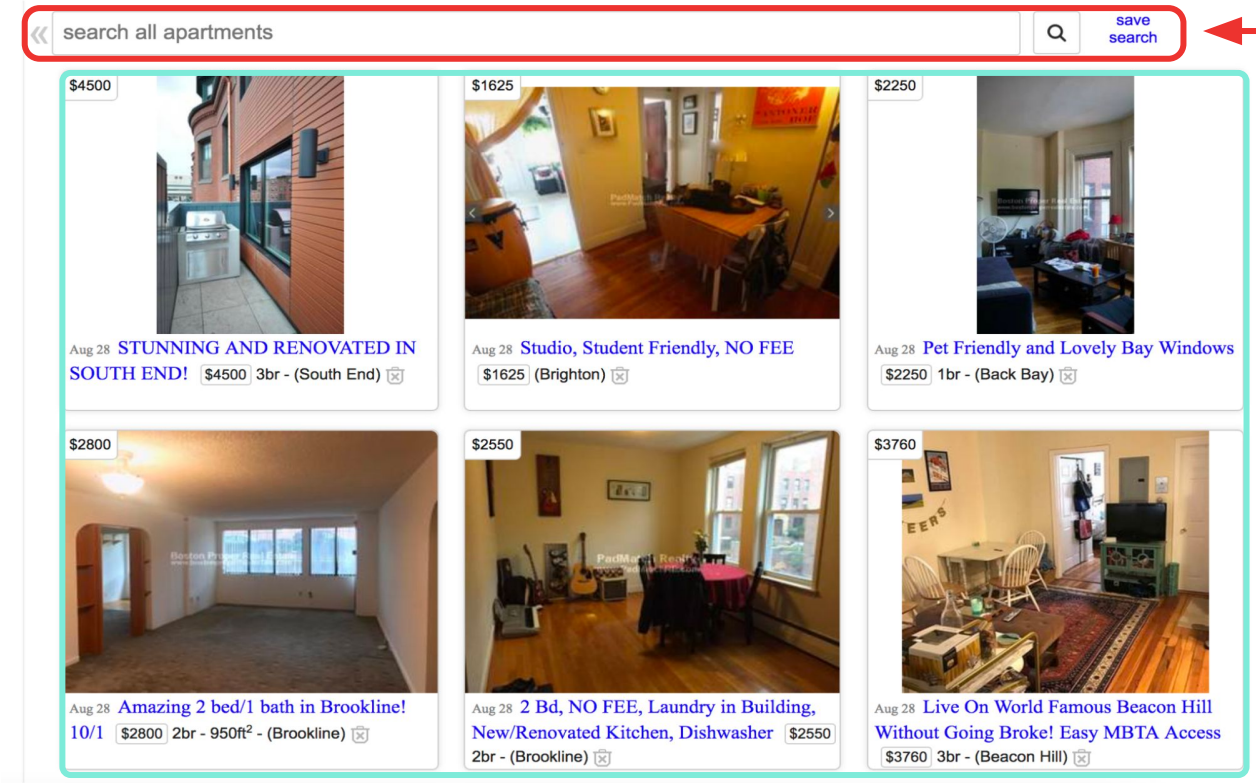

Component Anatomy

- A **component's anatomy** in React refers to the structure and organization of its code. It includes:
  - Props
  - State
  - Render method
  - Constructor
  - Event handlers
  - Hooks



Understanding the anatomy of a React component is essential for building and maintaining a React application.

Component Anatomy

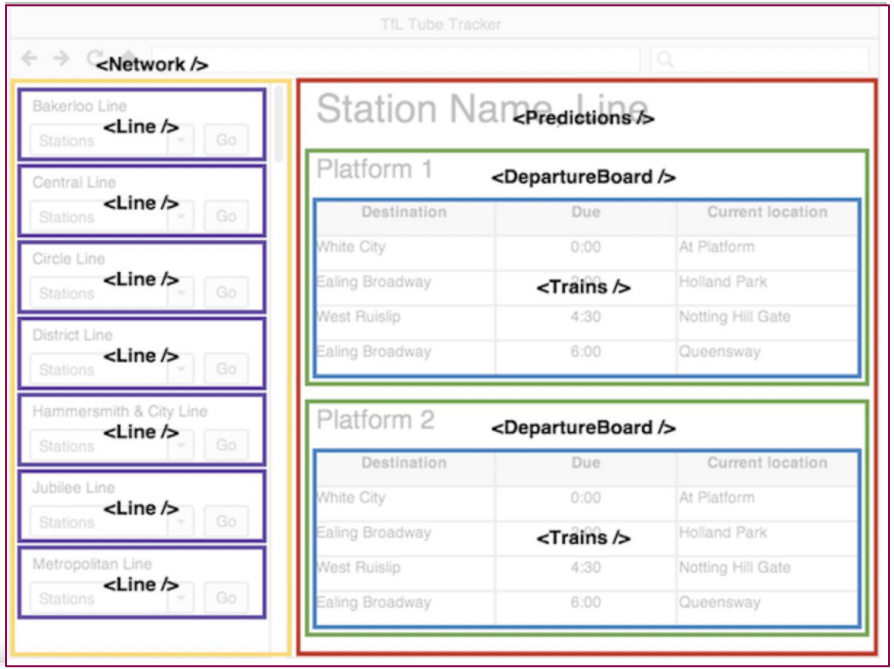


Navbar is it's own component.

Identical in structure but each has **different information** populating them.

Source: CraigsList

Discussion:
Component Anatomy



Let's look at this mock-up of a webpage. **What do you think are the components that make up this page?**

Challenge: *What are the total number of components on this page?*

Guided Walk-Through:

Component Anatomy

10 minutes

Code Along: A Very Basic Component

What does a component look like? Let's practice with a simple "Hello World" component.

In this section, we'll walk through:

- ➔ Removing the pre-filled contents of your hello-world app.
  - ◆ Create React App filled your app with sample content - let's make room for your app!
- ➔ Adding your own component definition.
  - ◆ You want the app to display the words **"Hello World!"**
- ➔ Going through what we've done in detail!

Break Time

Modern Engineering

React Props

Component Data With Props

The React framework was built to handle data that changes over time. In an upcoming exercise, we will learn how to display a greeting change dynamically based on the user's name.

The **unidirectional data flow** is one of React's most pleasant-to-work-with design decisions: data gets passed down from a parent component to its children components.

This creates a design split between **container components** that manage an app's data, and the **presentational components** that rely on props.

Discussion:

Passing Data Dynamically

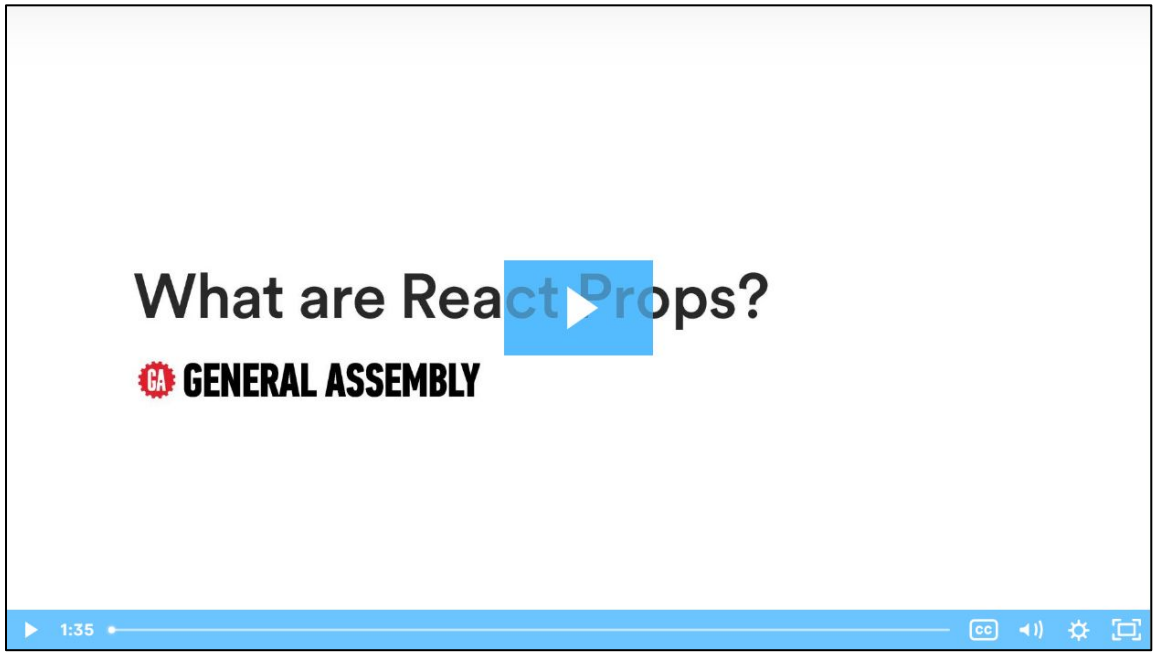
5 minutes

Why would passing data dynamically be a benefit when creating applications?

Come off mute and discuss!

Component Data With Props

The question is, how do we add a name to our Hello component without hardcoding it into the component's return method?



3

Let's watch a video

Find out! Try it yourself alongside this video in this codepen.

Guided Walk-Through:

Adding Props to Hello World

Let's extend our Hello World app to use props as a way of passing data into the presentational component that displays the user's name

Solo Exercise:

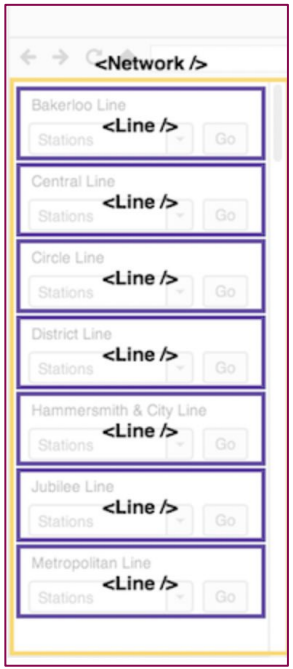
Subway Lines Code Along

15 minutes

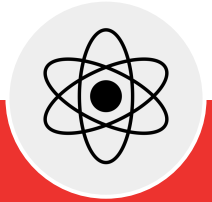
Let's recreate the component hierarchy from our subway lines example.

Each Line component will be given its **name** via a prop. Don't worry about the other aspects in the wireframe.

We'll hardcode the props for now, but an array of JSON objects is coming!



Of course, we often want components to display more complex information. To do so, we can pass multiple properties to our component. We'll use the same two steps we took to add the first prop.



Multiple Props

First, add another prop to the component call:

<Hello name={"Carl Sagan"} />, changes to <Hello name={"Carl Sagan"} age={62} />.

Update your index.js file to reflect this.

```
import * as React from "react";
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import "./style.css";

import Hello from "./App";

const rootElement = document.getElementById("root");
const root = createRoot(rootElement);

root.render(
    <StrictMode>
        <Hello name={"Carl Sagan"} age={62} />
    </StrictMode>
);
```


Multiple Props

Now, in our component definition we have access to both values.

The second step is to change the **Hello functional component** in App.js to use the age information.

```
import * as React from "react";
import "./style.css";

export default function Hello(props) {
    return (
        <div>
            <h1>Hello {props.name}!</h1>
            <p>You are {props.age} years old.</p>
        </div>
    );
}
```


Multiple Props...passed from an object

If we have many props, it might get difficult to keep track when we're passing everything in to render a component. A better practice is to organize values in some kind of object and then pass props to the component from that object.

Currently, in index.js, we put Carl Sagan's name and age directly into the root.render call.

Instead, we'll create an object that holds Carl Sagan's name and age, making it clearer for other developers and easier to change in the future. **In your index.js file, below the import statements, add this object definition**

```
const person = {
    personName: "Carl Sagan",
    personAge: 62,
};
```


Multiple Props...passed from an object

Next, we'll update what's passed into the component.

Near the bottom of your index.js, modify the root.render() call:

```
root.render(
    <StrictMode>
        <Hello name={person.personName} age={person.personAge} />
    </StrictMode>
);
```

Multiple Props...passed from an object

We don't have to change anything in App.js. **Why?**

- It's still receiving exactly the same values for exactly the same two props - name and age. We're just sending it those values in a slightly different way.

Check it out!
If you refresh your browser, nothing should have changed.

Try it out! Change the values inside the person object *without* changing the root.render() call.

Multiple Props...passed from a more complex object

Since we're just pulling props out of an object, we can use any object we want. For example, we can nest an array inside it.

Let's say our user has some favorite animals. **Update your object to include an array:**

```
const person = {
    personName: "Carl Sagan",
    personAge: 62,
    favorites: ["Birds", "Tigers", "Dinosaurs count!"],
};
```


Multiple Props...passed from a more complex object

We can use this new information as a prop, just like normal.

You could choose to pass a single element **(favorites[0])** or the entire array.

We'll use the entire array so that the component can display all a person's favorite animals.

First, update your root.render() call in index.js:

```
root.render(
    <StrictMode>
        <Hello
            name={person.personName}
            age={person.personAge}
            animals={person.favorites}
        />
    </StrictMode>
);
```


Multiple Props...passed from a more complex object

If you check your application now, nothing has changed.

Remember, a component class will just ignore any props it receives that it doesn't use. But, we want to use the favorite animals!

Second, update your Hello functional component's return method in App.js:

```
<div>
    <h1>Hello {props.name}!</h1>
    <p>You are {props.age} years old.</p>
    <p>You love: {props.animals}</p>
</div>
```


Multiple Props...passed from a more complex object

If you check the page now, you'll see React prints the entire array, as that's what was passed in.

If we wanted to include all the animals clearly, we could fix the spacing.

Instead, to review some syntax, **let's just modify the code to render the first value.**

```
<div>
    <h1>Hello {props.name}!</h1>
    <p>You are {props.age} years old.</p>
    <p>You love: {props.animals[0]}</p>
</div>
```


Solo Exercise:

Blog Post Props

15 minutes

Follow this readme for instructions on exercising props by creating a Blog Post component that displays its given props.

Lunch Time

React Foundations

Nesting Components

Looping and Nesting Components

A very common pattern in react is the need to loop over an array and display a presentational component for each item in the array.

With the way JSX works, this looping needs to create an array of components.

We'll need the **.map** method from JavaScript as a way of easily creating an array by applying a function across all items in an array.

Nested Components Example

```
const subwayLines = [
    { title: "Red Line"},
    { title: "Brown Line"},
    { title: "Blue Line"}
]

{
    subwayLines.map((line)=>{
        return <Line title={line.title}/>
    })
}
```

Guided Walk-Through:

Nesting Comments in a Post

45 minutes

Let's work through a nested components example using the Post and Comment components.

Partner Exercise:

Nesting Authors in a Post

30 minutes

Follow the instructions provided in this README file to practice adding another nested component into our Post component.

Solo Exercise:

Lord of the Rings Films Page

1 hour

Practice using nested components and props by creating a static site that displays information about several movies.

Solo Exercise:

Bonus - Favorite Songs App

1 hour

Practice using nested components and props by creating a static site that displays information about your favorite songs.

Recap

- Why use React?
- Components
- Props

Any Questions

Solo Exercise:

Daily Exit Ticket

3 minutes

- Please take a moment to give us your feedback after today's course!
- Use the QR code or follow the link: https://bit.ly/pruMEFc2

bitly

Scan Me!


GA